Methods:

Hooking into changes

Creating and changing classes is great, but intervening into the process of someone else's change is even cooler. So, in this video, we will cover different hook points that Ruby provides.

We will learn:

- Class/module extension hooks
  (included, extended, prepended, inherited)
- Method adding/changing hooks
  (method_added, method_removed, singleton_method_undefined, etc.)
- Method/constant not found
  (method_missing, const_missing)

Class/module related hooks

- included
- extended
- prepended
- inherited

included / extended / prepended

```ruby
# Ruby documentation:
#   http://ruby-doc.org/core/Module.html#method-i-included

module A
  def A.included(mod)
    puts "#{self} included in #{mod}"
  end
end

module Enumerable
  include A
end

# A included in Enumerable
```

```ruby
# Ruby documentation:
#   http://ruby-doc.org/core/Module.html#meth

module A
  def A.extended(mod)
    puts "#{self} extended in #{mod}"
  end
end

module Enumerable
  include A
end

# A extended in Enumerable
```

inherited

```ruby
# Ruby documentation:
#   http://ruby-doc.org/core/Class.html#method-i-inherited

class Foo
    def self.inherited(subclass)
        puts "New subclass: #{subclass}"
    end
end

class Bar < Foo
end

class Baz < Bar
end

# Produces:
> New subclass: Bar
> New subclass: Baize
```

Method adding/removing hooks

- method_added
- method_removed
- method_undefined
- singleton_method_added
- singleton_method_removed
- singleton_method_undefined

method_added

```ruby
# Ruby documentation:
#   http://ruby-doc.org/core/Module.html#method-i-method_added

module Chatty
    def before_method_added_method()
    end

    def self.method_added(method_name)
        puts "Adding #{method_name.inspect}"
    end

    def some_instance_method()
    end

    def self.some_class_method()
    end
end

# Produces:
> Adding :some_instance_method
```

method_removed / method_undefined

```ruby
# Ruby documentation:
#   http://ruby-doc.org/core/Module.html#method-i-method_removed

module Chatty
    def self.method_removed(method_name)
        puts "Removing #{method_name.inspect}"
    end

    def self.some_class_method
    end

    def some_instance_method
    end

    class << self
        remove_method :some_class_method
    end

    remove_method :some_instance_method
end

# Produces:
> Removing :some_instance_method
```

singleton_method_added / removed / undefined

```ruby
# Ruby documentation:
#   http://ruby-doc.org/core/Module.html

module Chatty
    def Chatty.singleton_method_added(id)
        puts "Adding #{id.id2name}"
    end

    def self.one()     end
    def two()          end
    def Chatty.three() end
end



# Produces:
> Adding singleton_method_added
> Adding one
> Adding three
```

```ruby
# Ruby documentation:
#   http://ruby-doc.org/core/Module.html#method-i-singleto

module Chatty
  def Chatty.singleton_method_removed(id)
    puts "Removing #{id.id2name}"
  end

  def self.one()     end
  def two()          end
  def Chatty.three() end

  class << self
    remove_method :three
    remove_method :one
  end
end

# Produces:
> Removing three
> Removing one
```

“ClassMethods” trick from Rails’ codebase

```ruby
module MyModule
  def self.included(base)
    base.extend ClassMethods
  end

  def instance_method() end

  module ClassMethods
    def class_method()  end
  end
end

class MyClass
    include MyModule
end

MyClass.class_method
MyClass.new.instance_method
```

```ruby
module MyModule
  def instance_method() end

  module ClassMethods
    def class_method()  end
  end
end

class MyClass
    include MyModule
    extend  MyModule::ClassMethods
end

MyClass.class_method
MyClass.new.instance_method
```

method_missing / const_missing

```ruby
# Ruby documentation:
#   http://ruby-doc.org/core/Module.html#method-i-const_missing

def Foo.const_missing(name)
  name                                  # return the constant name as Symbol
end

Foo::UNDEFINED_CONST                    # :UNDEFINED_CONST



module Foo
    def self.const_missing(name)
        const_set(name, Class.new)
    end
end

obj = Foo::Omg.new
obj.class                                # Foo::Omg
```